Overview:

1. What is Transaction?
2. Consideration for Isolation
3. JTS (Java Transaction Service)
4. JTA (Java Transaction API)
5. EJB and Transaction
6. Example
**What is Transaction?**

**Consider the following:** Online Banking (1)

Making $1,000 payment from your checking account

Your checking account:

- $1,000
- $1,000
- $1,000
- $0

Query → Pay $1,000 → Query

You have $1,000 → You have $0

---

**What is Transaction?** (cont.)

**Consider the following:** Online Banking (2)

What if somebody else wants to withdraw $700 at the same time?

Pay $700

- $1,000
- $1,000
- $1,000
- $0

Query → Pay $1,000 → Query

You have $1,000 → sorry... → You have $300
What is Transaction? (cont.)

Consider the following: Online Banking (3)
Let's close look at the task "payment". The worst case:

What is Transaction? (cont.)

• A transaction is a unit of work.
• A transaction is necessary mechanism for distributed Enterprise application.
• There are only two ways a transaction to terminate:
  • Commit
  • Rollback
• A transaction has the following (ACID) characteristics:
  • A transaction is Atomic; if interrupted by failure, all effects are undone (rolled back).
  • A transaction produces Consistent results; the effects of a transaction preserve invariant properties.
  • A transaction is Isolated; its intermediate states are not visible to other transactions. Transactions appear to execute serially, even if they are performed concurrently.
  • A transaction is Durable; the effects of a completed transaction are persistent; they are never lost (except in a catastrophic failure).
Consideration for Isolation

- **Dirty read**
  Reading the data that is used but not yet committed by other transaction.

- **Non-repeatable read**
  Reading the data that has been altered by other transaction since the last read.

- **Phantom read**
  Reading the newly created data by other transaction.

JTS (Java Transaction Service)

- Specifies the low-level implementation of a Transaction Manager.
- Implements the Java mapping of the OMG Object Transaction Service (OTS) 1.1 specification.
- Uses the CORBA ORB/TS interfaces and IIOP (Internet Inter ORB Protocol).
- The packages `org.omg.CosTransactions` and `org.omg.CosTSPortability`
- Supports JTA (Java Transaction API)
JTA (Java Transaction API)

- Interfaces between a transaction manager and the parties involved in a distributed transaction system including:
  - Resource manager
  - Application server
  - Transactional applications.

- Implementation independent

- Protocol independent

- The packages `javax.transaction`

JTA (Java Transaction API) (cont.)

**Question:**

How much should I know about JTS/JTA?

**Answer:**

You don't need to know too much detail about JTS/JTA to take advantage of the transaction unless you are writing something like a JDBC driver. For example, using transaction with Enterprise JavaBeans requires you to implement only one interface from JTA.
**EJB and Transaction ("UserTransaction" interface)**

**Interface:**
public interface javax.transaction.UserTransaction

**Methods:**
Only six methods: `begin()`, `commit()`, `getStatus()`, `rollback()`, `setRollbackOnly()`, and `setTransactionTimeout(int)`.

**Classes:**
- public class javax.transaction.HeuristicMixedException
- public class javax.transaction.HeuristicRollbackException
- public class javax.transaction.NotSupportedException
- public class javax.transaction.RollbackException
- public class javax.transaction.SystemException

**EJB and Transaction (pseudo code)**

```java
try {
    trans.begin();
    :
    : Do the work
    :
    trans.commit(); // Everything went fine. Let's commit
} catch (Exception e) {
    trans.rollback(); // An exception thrown!! Let's rollback.
}
```
EJB and Transaction (Transactional Option)

Developer sets a transaction option at deployment time. Options are:

- NotSupported
- Supports
- Required
- RequiresNew
- Mandatory

EJB and Transaction (Transaction Isolation level)

Developer sets a transaction option at deployment time. Options are:

- TRANSACTION_READ_UNCOMMITTED
  Least strict. Changes made by other transaction are immediately available before commit. There is a risk the data may rollback.

- TRANSACTION_READ_COMMITTED
  No dirty-read allowed. Can see all committed changes (non-repeatable-read and phantom-read is ok).

- TRANSACTION_REPEATABLE_READ
  No dirty read. No repeatable read. Phantom-read is ok (it is ok to read newly inserted rows as long as they are committed).

- TRANSACTION_SERIALIZABLE
  Most strict isolation. Locks all data until it commits. No dirty read allowed. No repeatable read. No Phantom-read.
EJB and Transaction (methods)

void begin()
Create a new transaction and associate it with the current thread.

void commit()
Complete the transaction associated with the current thread.

int getStatus()
Obtain the status of the transaction associated with the current thread.

void rollback()
Roll back the transaction associated with the current thread.

void setRollbackOnly()
Modify the transaction associated with the current thread such that the only possible outcome of the transaction is to roll back the transaction.

void setTransactionTimeout(int seconds)
Modify the timeout value that is associated with transactions started by subsequent invocations of the begin method.

EJB and Transaction (Exceptions)

NotSupportedException
Thrown if the thread is already associated with a transaction and the Transaction Manager implementation does not support nested transactions.

RollbackException
Thrown to indicate that the transaction has been rolled back rather than committed.

HeuristicMixedException
Thrown to indicate that a heuristic decision was made and that some relevant updates have been committed while others have been rolled back.

HeuristicRollbackException
Thrown to indicate that a heuristic decision was made and that all relevant updates have been rolled back.

SystemException
Thrown if the transaction manager encounters an unexpected error condition.
/*
 * Copyright 2001 Sun Microsystems, Inc. All Rights Reserved.
 * This software is the proprietary information of Sun Microsystems, Inc.
 * Use is subject to license terms.
 */

import java.util.*;
import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import javax.transaction.*;

public class TellerBean implements SessionBean {
    private String customerId;
    private double machineBalance;
    private SessionContext context;
    private Connection con;
    private String dbName = "java:comp/env/jdbc/TellerDB";

    public void withdrawCash(double amount) {
        UserTransaction ut = context.getUserTransaction();
        try {
            ut.begin();
            updateChecking(amount);
            machineBalance -= amount;
            insertMachine(machineBalance);
            ut.commit();
        } catch (Exception ex) {
            try {
                ut.rollback();
            } catch (SystemException syex) {
                throw new EJBException("Rollback failed: " + syex.getMessage());
            }
            throw new EJBException("Transaction failed: " + ex.getMessage());
        }
    }

    public double getCheckingBalance() {
        try {
            return selectChecking();
        } catch (SQLException ex) {
            throw new EJBException("Unable to get balance: " + ex.getMessage());
        }
    }

    public void ejbCreate(String id) throws CreateException {
        customerId = id;
        try {
            makeConnection();
            machineBalance = selectMachine();
        } catch (Exception ex) {
            throw new CreateException(ex.getMessage());
        }
    }

    public void ejbRemove() {
        try {
            con.close();
        } catch (SQLException ex) {
            throw new EJBException(ex.getMessage());
        }
    }

    public void ejbActivate() {
        try {
            makeConnection();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }
}
public void ejbPassivate() {
    try {
        con.close();
    } catch (SQLException ex) {
        throw new EJBException(ex.getMessage());
    }
}

public void setSessionContext(SessionContext context) {
    this.context = context;
}

public TellerBean() {}
private double selectChecking() throws SQLException {

    String selectStatement =
        "select balance " +
        "from checking " +
        "where id = ?";
    PreparedStatement prepStmt =
        con.prepareStatement(selectStatement);
    prepStmt.setString(1, customerId);
    ResultSet rs = prepStmt.executeQuery();
    if (rs.next()) {
        double result = rs.getDouble(1);
        prepStmt.close();
        return result;
    } else {
        prepStmt.close();
        throw new EJBException
            ("Row for id " + customerId + " not found.");
    }
} // TellerBean